VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Hra Finanční svoboda pro platformu Android a desktop

Game Financial Freedom for Platform Android and Desktop

2014 Lukáš Vlček

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Vlček**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Hra Finanční svoboda pro platformu Android a desktop
Game Financial Freedom for Platform Android and Desktop

Zásady pro vypracování:

Cílem práce je vytvořit elektronickou verzi hry Finanční svoboda, kterou bude možno spustit jak na platformě android tak na platformě Java SE na běžném počítači. Hra bude navržena tak aby umožňovala hru více hráčů, díky připojení k serveru.

Hra umožní:
1. Hraní hry více hráčů v prostředí internetu.
2. Využití hráče s jednoduchou umělou inteligencí.
3. Přidat rozšíření, která přidávají do hry nové prvky finančního trhu nebo upravují chování jednotlivých prvků hry.
4. Samostatná hra bez připojení k serveru.
5. Program umožní připojení do současně vyvíjeného portálů her v jazyce Java.

Práce bude obsahovat:
1. Popis použitých technologií.
2. Implementaci výše popsaného programu (serverové části, klienta pro platformu Android a klienta pro desktop).
3. Programátorskou dokumentaci řešení s využitím diagramů jazyka UML.

Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
[2] DARWIN, Ian F. Java cookbook. 2nd ed. Sebastopol, CA: O'Reilly, c2004, xxiv, 829 p. ISBN 05-960-0701-9. Dostupné z: http://it-ebooks.info/book/2249/

Dále podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2014

.....

Rád bych na tomto místě poděkoval Ing. Davidu Ježkovi, Ph.D. za jeho cenné rady při vývoji aplikace a dále své rodině za poskytnutí podmínek pro vznik této práce.

Abstrakt

Cílem bakalářské práce je vytvořit elektronickou verzi hry Finanční svoboda, kterou bude možné spustit jak na platformě Android, tak na platformě Java SE na běžném počítači. Práce je rozdělená do tří částí. V první části se věnuji pravidlům dané hry, ve druhé použitým technologiím při vývoji aplikace a ve třetí části se věnuji samotnému řešení a tvorbě aplikace.

Klíčová slova: Java, Android, JAXB, finance, LWJGL, grafická knihovna, cashflow, pojištění, investice, půjčka

Abstract

Goal of this Bachelor Thesis is to create a virtual version of the Financial Freedom desk game for both Java Desktop and Android platforms. My thesis is divided into three separate parts. The first one explains the rules of the game, second part is dedicated to technologies used to develop the application and the last part describes the actual development of the application.

Keywords: Java, Android, JAXB, finances, LWJGL, graphics library, cashflow, insurance, investment, loan

Seznam použitých zkratek a symbolů

XML	–	eXtensible Markup Language
HTML	–	Hypertext Markup Language
OpenGL	–	Open Graphics Library
OpenCL	–	Open Computing Language
OpenAL	–	Open Audio Library
LWJGL	–	LightWeight Java Game Library
GUI	–	Graphical User Interface
WORA	–	Write Once, Run Anywhere
JVM	–	Java Vritual Machine
TCP	–	Transmission Control Protocol
UDP	–	Universal Datagram Protocol

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Cílem této bakalářské práce je navrhnout a implementovat elektronickou verzi deskové hry Finanční svoboda.

Praktická část se skládá ze čtyř částí. V první části je cílem vytvořit obecnou knihovnu, která bude obsahovat všechny potřebné třídy a funkce pro napojení na GUI daného klienta, bude obsahovat veškerou logiku a pravidla hry a rozhraní pro napojení Event Listenerů pro komunikaci knihovny s grafickým rozhraním aplikace. Dále bude knihovna obsahovat všechny typy karet aktivit a událostí. Knihovna bude podporovat jak hru jednoho hráče s umělou inteligencí, tak hru přes internet. Hru bude možné rozšířit o další prvky, které budou ovlivňovat množství karet ve hře nebo chování hry v různých stavech.

Druhou částí práce je implementace již zmíněné knihovny a vytvoření herního internetového serveru. Pro vytvoření hry bude potřeba poslat požadovaný název hry a požadovanou úroveň obtížnosti.

Třetí částí práce je vytvoření desktopové aplikace, která bude podporovat hraní v módu Single Player s umělou inteligencí a bude se schopna připojit na již zmíněný server. Grafické rozhraní bude zobrazovat všechny důležité informace o stavu hry, jakými jsou například aktuální kolo hry, čas do konce kola, informace o všech hráčích, informace o daném klientovi, jeho cashflow, aktuální finanční situace včetně situace na finančním trhu. Dále bude možné v grafické rozhraní nakupovat a prodávat karty, případně hru předčasně ukončit.

Čtvrtou části práce je vytvoření portu této aplikace pro mobilní zařízení. Cílem této části je zachování co nejvíce podobného grafického rozhraní s důrazem na zachování komfortu při hraní hry na menší obrazovce. Na display mobilního telefonu nebudou zobrazeny všechny informace o stavu aktuální hry.

Textová část práce je rozdělena do tří částí. První část je věnována pravidlům hry Finanční svobody. Vysvětluje potřebná pravidla hry, její začátek, průběh celé hry i jednoho kola hry včetně popisu chování jednotlivých karet aktivit a karet událostí. Druhá část je věnována popisu použitých technologií a ukazuje na jednoduchých příkladech, jakým způsobem lze jednotlivé technologie implementovat v jazyce Java. V poslední části bude popis implementace všech již zmíněných částí aplikace, popis grafických rozhraní aplikací a dokumentace k rozšíření hry o další prvky.

2 Pravidla hry

V následující kapitole si popíšeme pravidla online hry Finanční svoboda, která vychází ze stejnojmenné deskové hry, avšak v určitých oblastech chování hry upravuje.

2.1 Začátek hry

Hráč dostane na začátku hry přiděleného klienta a jeho rodinu, kterou má za úkol vyvést z finančních obtíží. Každý klient má své specifické finanční problémy a různé rodinné zázemí a životní cíle. Rodina se skládá z otce, matky a jednoho dítěte. Finanční situace klienta je charakterizována množstvím úspor, finančním příjmem rodiny, jejich výdaji, aktuálními dluhy a životními cíli.

Na začátku hry je klientovi 30 let. První kolo má sloužit k seznámení se s klientem a prozkoumání jeho životní situace. Hráč vytvoří strategii, jak vyvést klienta z jeho tíživé situace a zajistit mu finanční stabilitu. První z cílů musí být splněn do 39. roku života vašeho klienta, tedy třetího kola hry. Druhý z cílů musí být splněn do 48. roku života vašeho klienta, tedy šestého kola hry. Pokud na splnění cíle nemá klient dostatečné množství financí, bude nucen si vzít půjčku.

2.2 Průběh hry

Hra se skládá z 10 kol. V každém kole může hráč provádět transakce s majtekem svého klienta. K tomu, aby dosáhl cílů svých klientů může využít všech možných prostředků finančního trhu, které hra nabízí. V 39. a 48. roku, tedy ve 3. a 6. kole je hráč povinen splnit svému klientovi jeho životní cíl.

Prvním cílem je pořídit klientovi vlastní bydlení. Splněním tohoto cíle se klientovi uvolní finanční prostředky a zvýší cashflow, jelikož nebude muset nadále platit nájemné.

Druhý cíl souvisí se zlepšením životního komfortu klienta a jedná se zejména o nákup rekreační chaty, nákup automobilu či prožití vysněné dovolené. Druhý cíl je jednorázovou investicí klienta a neovlivňuje narozdíl od prvního cíle další atributy hry.

V případě, kdy hráč nemá dostatek peněz pro splnění cíle se musí zadlužit. Pokud se jedná o zajištění vlastního bydlení, přidá se klientovi karta hypotéka. V druhém případě či obecném nedostatku financí způsobeným především záporným cashflow, bude klientovi přidaná karta půjčka.

2.3 Průběh kola

Každé kolo se skládá ze tří částí. V první části se vygeneruje pohyb investičních prvků pro dané období, který ovlivní jejich nákupní i prodejní cenu. Pohyb podílových listů OPF, dluhopisů i akcií je náhodný, přesto závislý jeden na druhém. Nemůže se tedy stát, že při stoupání ceny dluhopisů bude klesat hodnota podílových listů.

V další části hry si každý hráč vytáhne z balíku jednu kartu události. Karta události pozitivně i negativně upravuje životní situaci klienta.

Poté má každý hráč 2 minuty na to, aby provedl všechny finanční operace tak, aby postupně zlepšoval finanční situaci klienta. Po dvou minutách je kolo ukončeno a probíhá stejným způsobem kolo další.

2.4 Prvky finančního trhu

V následující kapitole si popíšeme jednotlivé finanční prvky, které ve hře Finanční Svoboda najdeme.

2.4.1 Investiční karty

Investiční karty jsou vhodným způsobem investice volných finančních prostředků rodiny. Hodnota těchto karet se v každém kole mění a jsou ovlivňovány náhodnými událostmi hry. Hra Finanční svoboda nabízí tři druhy možných investic.

Akcie jsou nejvýnosnější možnou investicí peněz ve hře, ale narozdíl od dluhopisů a podílových listů OPF jsou více ovlivňovány náhodnými událostmi.

Dluhopisy jsou méně výnosné oproti akciím, nicméně zvyšování jejich hodnoty v čase je stabilnější a méně ovlivňováno náhodnými událostmi hry.

Podílové listy jsou méně výnosné než akcie. Jejich hodnota se zpravidla pohybuje opačným směrem oproti dluhopisům a akciím. Nákup podílových listů je tedy vhodný v době, kdy finanční trh prochází krizí.

2.5 Karty pojištění

Karta pojištění slouží k zajištění stability klienta v nejrůznějších životních situacích. Tyto situace jsou blíže popsány v kapitole 2.7.

2.5.1 Rizikové životní pojištění

Rizikové životní pojištění slouží k pojištění proti úrazu s následkem smrti klienta. V takovém případě přichází rodina o značnou část příjmů. Jedna karta představuje pojištění ve výši 500 000 Kč a měsíční poplatek ve výši 500 Kč. Lze vlastnit více než jednu kartu tohoto typu.

2.5.2 Kapitálové životní pojištění

Kapitálové životní pojištění také slouží k pojištění proti úrazu s následkem smrti klienta. Na rozdíl od rizikového životního pojištění je v tomto případě pojistná hodnota nižší. Oproti rizikovému živnotnímu pojištění, v případě nevyužití této karty, se v 60 letech či při zrušení karty část peněz vrací do rodinného rozpočtu. Tato karta tedy představuje nejen formu životního pojištění, ale také spoření. Výše pojistného vzhledem k věku klienta je blíže popsána v tabulce 1. Měsíční poplatek za tuto kartu činí 500 Kč. Lze vlastnit více než jednu kartu tohoto typu.

Věk klienta	Výše pojistného
30	200 000 Kč
33	174 000 Kč
36	150 000 Kč
39	126 000 Kč
42	105 000 Kč
45	86 000 Kč
48	67 000 Kč
51	50 000 Kč
54	33 000 Kč
57	13 000 Kč

Tabulka 1: Výše pojistného u karty kapitálové životní pojištění v závislosti na věku klienta

Věk klienta	Výše pojistného
30	300 000 Kč
33	262 000 Kč
36	226 000 Kč
39	192 000 Kč
42	160 000 Kč
45	130 000 Kč
48	101 000 Kč
51	74 000 Kč
54	48 000 Kč
57	24 000 Kč

Tabulka 2: Výše pojistného u karty penzijní připojištění v závislosti na věku klienta

2.5.3 Penzijní připojištění

Penzijní pojištění je karta, která stojí na rozmezí mezi kartou pojištění a investiční kartou. Penzijní připojištění lze uzavřít kdykoliv v průběhu hry a je standardně vypláceno v 60 letech. Pojistná částka se liší v závislosti na věku klienta při sjednání pojištění. Detailně je popsána v tabulce 1. Měsíční poplatek za tuto kartu činí 500 Kč. Lze vlastnit více než jednu kartu tohoto typu.

2.5.4 Stavební spoření

Stavební spoření představuje formu krátkodobé investice, je sjednáno na 6 let a na rozdíl od ostatních karet nelze tuto kartu předčasně zrušit. Měsíční poplatek činí 1750 Kč a za 6 let je klientovi vyplcena částka 140 000 Kč. Lze vlastnit více než jednu kartu tohoto typu.

2.5.5 Pojištění majetku

Pojištění majetku ochraňuje klienta v případě, kdy dojde ke škodě na majetku i domácnosti, a to vlastním i cizím. Měsíční poplatek činí 250 Kč a jedna karta kryje 100% škody na majetku jedné nemovitosti. Lze vlastnit více než jednu kartu.

2.5.6 Pojištění trvalých následků úrazu

Pojištění trvalých následků úrazu ochraňuje klienta v případě, kdy klient trvale ztratí možnost pracovat následky úrazu. Měsíční poplatek činí 250 Kč za pojistnou částku 1 000 000 Kč. Lze vlastnit více než jednu kartu.

2.6 Karty půjčky

2.6.1 Hypotéka

Hypotéka je forma půjčky uzavřená za účelem stavby či nákupu bytu či domu pro klienta. Hypotéka je uzavírána na 15 let, měsíční splátka činí 2 000 Kč a je uzavřena ve výši 500 000 Kč. Karty může být splacene předem. Cena za předčasné splacení se mění v závislosti na počtu odehraných kol od nákupu karty. Lze vlastnit více než jednu kartu tohoto typu

2.6.2 Spotřebitelský úvěr

Spotřebitelský úvěr je forma půjčky, jejíž účel nemusí být blíže specifikován. Měsíční poplatek za tuto kartu činí 500 Kč, výše úvéru je 50 000 Kč, což odpovídá reálné úrokové sazbě RPSN 12% p.a.. Karta je hrou využívána automaticky v případě nedostatku financí způsobené událostmi nebo negativním cashflow.

2.7 Události

V následující kapitole si popíšeme všechny náhodné události, které mohou ve hře nastat.

2.7.1 Škoda na majetku

Karta škoda na majetku představuje finanční výdaje klienta, které musí být uhrazeny okamžitě. V případě nedostatku financí je nutné využít kartu spotřebitelský úvěr detailně popsanou v kapitole 2.6.2. Pokud hráč vlastní kartu pojištění majektu popsanou v kapitole 2.5.5, nemá tato karta žádný účinek.

2.7.2 Narození dítěte

Vašemu klientovi se narodilo dítě, které negativně ovlivní náklady rodiny. Ve výjimečných případech se mohou narodit dvojčata. Počet dětí v jedné rodině je omezen na tři. Pokud má rodina již tři děti a klient si tuto kartu vytáhne, nemá žádný účinek.

2.7.3 Smrt v rodině

Klientovi zemřel jeden z členů rodiny, což negativně ovlivní jeho měsíční příjem. V případě, že měl klient sjednáno životní pojištění, dojde k vyplacení této částky.

2.7.4 Výhra, ztráta peněz

Karta pozitivně i negativně ovlivňuje množství peněz klienta. Jedná se o jednorázový zisk či ztrátu peněz, který dále nijak neovlivňuje cashflow klienta.

2.7.5 Ztráta zaměstnání

Klient ztratil zaměstnání a trvalo mu 6 měsíců než si našel novou práci. V tomto období nepobíral žádný plat a to v daném období negativně ovlivnilo cashflow rodiny.

2.7.6 Invalidita

Klient se zranil a v důsledku tohoto zranění nemůže dále pracovat a pobírat plat. Pokud klient vlastní kartu pojištění trvalých následků úrazu, popsanou v kapitole 2.5.6, bude mu vyplacena suma ve výši 100% pojistné částky.

2.8 Konec hry a hodnocení

Hra končí v 60 letech, tedy po 10 herních kolech. V tomto kole jsou automaticky prodány všechny investiční karty klienta a je určeno finální skóre klienta, podle kterého se určí vítěz hry.

Vítězem se stává hráč s nejvyšším skóre, které je určeno podle splněných cílů a dosažené finanční nezávislosti. Za každý splněný cíl hráč obdrží 300 bodů. Počet bodů za dosaženou finanční nezávislost je určen podle dosažení požadované finanční nezávislosti, která je pro každého klienta specifická.

3 Přehled použitých technologií

3.1 Java

Java je hybridní objektově orientovaný jazyk, jehož počátky spadají do 90. let, konkrétně do června 1991. V tomto roce skupina programátorů pod vedením Jamese Goslinga pracovala na programovacím jazyku určeném především pro televize a ostatní embeded systémy. Cílem jeho týmu bylo vyvinout jazyk s podobným stylem zápisu jako jazyk C/C++.

Původně se jazyk měl jmenovat Oak, podle dubu, který stál poblíž Goslingovy kanceláře. Jelikož ale takový programovací jazyk už existoval, byl vybrán nejprve název Green a poté již dnes známý název Java, který označuje druh kávy, kterou Goslingův tým v dané době ve velkém množství konzumoval.

Uvedení na trh se Java dočkala v roce 1995. Java se postupně začala stávat populárnější, jelikož prohlížeče brzo implementovaly tento jazyk ve formě spustitelných Java appletů. O tři roky později, v roce 1998, vyšla druhá verze Javy označovaná Java 1.2, která vyšla oproti svému předchůdci v několika verzích. Jave Enterprise Edition, která byla zaměřená především pro náročné podnikové aplikace, Java Standard Edition, která sloužila pro vývoj běžných aplikací a Java Mobile Edition, která díky použití pouze na mobilních platformách mohla být minimalizovaná o nepotřebné funkce. Další verze postupně rozšiřovaly funkce jazyka.

V letech 2006 až 2007 byla Java postupně uvolňována jako open-source až na drobné části kódu, ke kterým Sun Microsystems neměl vlastnické právo. V roce 2009 proběhla akvizice firmy Sun společností Oracle Corporation a v zápětí vývojový tým Javy opustil její autor James Gosling.

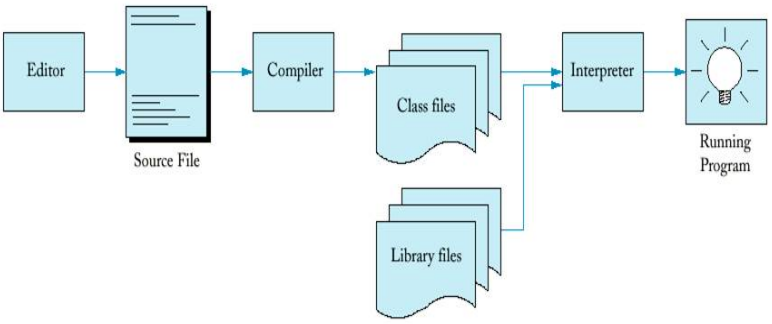
Java dodržuje standard WORA, čímž usnadňuje práci vývojářům, jelikož zdrojový kód lze spustit na jakémkoliv systému, který má nainstalovaný JVM. Java je objektově orientovaný jazyk, ale obsahuje také implementaci primitivních datových typů (Integer, String, Long, Byte, Double, Float, Short). Na rozdíl od jazyka C/C++ nepodporuje vícenásobnou dědičnost, ale dovoluje implementovat více rozhraní.

Java je jednoduchý a intuitivní jazyk, který striktně dodržuje své standardy, a tak je vývoj v Javě pro začínající programátory snadnější v porovnání s ostatními jazyky. O správu paměti se stará Garbage Collector, v Javě tedy nenajdeme funkce pro alokaci paměti a její následné uvolnění. Garbage Collector uvolní místo v paměti automaticky v případě, že na místo již neexistuje v aplikaci žádná reference. Průběh kompilace kódu je vidět na obrázku 1.

Nevýhodou Javy oproti jazyku C/C++ je jeho menší výkonnost a vyšší datová náročnost, jelikož samotný program se zkompiluje pouze do Java Byte kódu, který je následně intepretován JVM.

V době psaní práce je nejnovější verzí Javy verze 1.8, pro veřejnost označovaná jako Java Standard Edition 8. Popularitě jazyka pomohl také upravený JVM zvaný DalvikVM, který je podporován na mobilních zařízeních s operačním systémem Android.

Detailnější informace leze najít v referencích [2, 4].



Obrázek 1: Průběh kompilace zdrojového kódu jazyka Java

3.2 OpenGL

OpenGL [7] je multiplatformní open-source grafická knihovna poskytující aplikační rozhraní pro práci s 2D a 3D objekty. OpenGL, nebo jakákoliv její nadstavba, je využívána pro renderování objektů ve většině her.

Vývoj OpenGL začal v útrobách společnosti Silicon Graphics Inc. v roce 1991 a prvního vydání se dočkal o rok později. V té době nebylo žádné ucelené řešení pro práci s 2D a 3D objekty, a tak si každý programátor musel napsat svoji grafickou knihovnu, což bylo velmi neefektivní. Největším konkurentem se od roku 1995 stala knihovna Direct3D společnosti Microsoft, která je součástí balíčku DirectX.

3.3 LWJGL

LWJGL [6] je 2D i 3D grafická knihovna, která slouží k tvorbě her. LWJGL obaluje OpenGL, OpenCL a OpenAL. Poskytuje tenké API a umožňuje vývojářům přistupovat k jazyku Java nedostupným či špatně implementovaným prostředkům. Cílem LWJGL není zjednodušit vývoj aplikací, ale poskytnout přístup ke všem prostředkům systému. Kvůli tomu vzniklo hodně nadstaveb LWJGL.

3.4 Slick2D

Slick2D je multiplatformní knihovna sloužící k vývoji 2D her a je nadstavbou již zmíněné knihovny LWJGL. Slick2D zjednodušuje vývoj her a implementaci procesů, jakými je herní smyčka, renderování či vytvoření stavové hry. Knihovna je vhodná pro začínající vývojáře, jelikož její použití je velice jednoduché.

Součástí Slick2D jsou nativní knihovny pro Windows, Mac OS X a Linux.

Slick2D podporuje dva druhy her:

- BasicGame
- StateBaseGame

`BasicGame` je typ hry, který podporuje pouze jeden stav, tedy jednu obrazovku. Zatímco pro `StateBaseGame` lze lze definovat více stavů, tedy obrazovek, jako například hlavní menu, nápovědu a samotné herní okno. Mezi stavy se pak lze přepínat pomocí metody `enterState()`, která jako parametr přijímá ID daného stavu. To definujeme v těle dané třídy a získává se pomocí metody `getStateId()`.

3.4.1 Vytvoření stavové hry

Pro vytvoření stavové hry vytvoříme třídu, která je potomkem třídy `StateBasedGame`. Uvnitř metody `initStatesList()` přidáme pomocí metody `addState()` všechny potřebné instance stavů.

Ukázkové vytvoření instance hry je zobrazeno ve výpisu 1.

```
public class ExampleStateBasedGame extends StateBasedGame
{
    public ExampleStateBasedGame(String name)
    {
        super(name);
    }

    @Override
    public void  initStatesList (GameContainer gc) throws SlickException
    {
        addState(new SplashState());
        addState(new MainMenuState());
        addState(new GameState());
    }
}
```

Výpis 1: Vytvoření hry s několika stavy

3.5 NiftyGUI

NiftyGUI je grafická knihovna, která slouží k vytváření uživatelského rozhraní pro hry či jednoduché aplikace. Nifty spolupracuje s dalšími grafickými knihovnami, například s `LWJGL`, `LibGDX`, `JOGL` či již zmíněným `Slick2D`.

Rozhraní aplikace se dá vytvářet pomocí tzv. `BuilderClass`, které Nifty obsahuje nebo pomocí externího XML Souboru [8].

Nifty obaluje část Slick2D, a tak pro vytvoření hry je třeba použít jednu z následujících tříd.

- NiftyBasicGame
- NiftyStateBasedGame
- NiftyOverlayBasicGame
- NiftyOverlayGame

Pro vytvoření stavu je nutné použít jednu z následujících tříd.

- NiftyGameState
- NiftyBasicGameState
- NiftyOverlayGameState
- NiftyOverlayBasicGameState

3.5.1 Načtení dat z XML

```
protected void prepareNifty(Nifty nifty, StateBasedGame sbg)
{
    nifty.loadStyleFile("nifty-default-styles.xml");
    nifty.loadControlFile("nifty-default-controls.xml");
    nifty.fromXml("example/file.xml", "startingScreenId");
}
```

Výpis 2: Ukázka vytvoření NiftyGUI ve Slick2D pomocí XML

3.5.2 Tvorba GUI uvnitř programu

```
@Override
protected void prepareNifty(Nifty nifty, StateBasedGame sbg)
{
    nifty.loadStyleFile("nifty-default-styles.xml");
    nifty.loadControlFile("nifty-default-controls.xml");
    nifty.fromXml("example/file.xml", "startingScreenId");

    Screen s~= nifty.getCurrentScreen();
    s~= new ScreenBuilder("lobby") {{
        controller(new GameController());
        layer(new LayerBuilder("first") {{
            childLayoutAbsolute();

            panel(new PanelBuilder("topPanel") {{
                childLayoutAbsolute();
                x("10px");
                y("10px");

                control(new LabelBuilder("title") {{
                    text("Title");
                    color(Color.WHITE);
                    style("nifty-label");
                    marginBottom("10px");
                }});

            }});

            panel(new PanelBuilder("bottomPanel") {{
```

```
                x("10px");
                y("560px");
                childLayoutHorizontal();

                control(new ButtonBuilder("loanCardButton") {{
                    label("Buy␣Loan␣Card");
                    width("150px");
                    marginRight("10px");
                    height("30px");
                    interactOnClick("onBuyLoanCardClick()");
                }});
            }});
        }});
    }.build(this.nifty);

    nifty.gotoScreen(s.getScreenId());
}
```

Výpis 3: Ukázka vytvoření NiftyGUI ve Slick2D pomocí Class Builderů

3.6 Kryonet

Kryonet [10] je open-source knihovnou sloužící k vytvoření komunikace mezi serverem a klienty. Umožňuje spojení jak pomocí TCP, tak pomocí UDP. Kryonet je tedy vhodné řešení nejen pro aplikace s důrazem na kvalitu spojení, ale také real-time aplikace či hry.

Výhodou Kryonetu je jeho vysokoúrovňové API, které ulehčuje práci vývojářům, ale snižuje flexibilitu programu.

Pro serializace objektů a posílání dat po síti využívá Kryonet serializační knihovnu Kryo.

3.6.1 Kryo

Kryo [11] je serializační knihovna, která je součástí balíčku `Kryonet`.

Kryo již v základu umí serializovat následující objekty:

- boolean, Boolean
- byte, Byte, byte[]
- char, Character
- short, Short
- int, Integer, BigInteger, BigDecimal
- long, Long
- float, Float

- double, Double
- Collection, Map
- Date, TimeZone, Calendar, Currency
- Enum, EnumSet
- KryoSerializable

3.6.1.1 Vytvoření vlastního serializátoru Pokud chceme danou třídu serializovat pomocí vlastního serializátoru, musí tato třída implementovat rozhraní `KryoSerializable`, tedy metodu `read()` a `write()`.

```
public class SomeClass implements KryoSerializable
{
    public void write (Kryo kryo, Output output)
    {

    }

    public void read (Kryo kryo, Input input)
    {

    }
}
```

Výpis 4: Vlastní serializátor

3.6.2 Ukázka vytvoření instance Kryonet Serveru

Pro vytvoření instance serveru použijeme třídu `Server`. Pomocí metody `getKryo()` získáme objekt serializátoru `Kryo` a zaregistrujeme všechny třídy, které budeme posílat. Dále je třeba nabindovat server na určitý port a zapnout server pomocí metody `start()`. Pro zpracování příchozí komunikace můžeme pomocí metody `addListener()` přidat vlastního listenera. Ten je potomkem třídy `Listener` ze stejného balíčku a implementujeme metody `idle()`, která definuje chování při nečinosti spojení, metodu `received()`, která definuje chování při přijmutí dat, metodu `connected()`, která definuje chování při připojení klienta k serveru a metodu `disconected()`, která definuje chování při ztrátě spojení, která může nastat neočekávanou chybou na straně klienta nebo vypršením daného časového intervalu.

```
public class KryonetServerExample
{
    private Server server;
    private static final int PORT = 54345;

    public startServer() throws IOException
    {
        server = new Server();
```

```
        Kryo k~= server.getKryo();
        k.register(PlayerGameEnterAttemptPacket.class);
        k.register(PlayerGameEnterResponsePacket.class);
        k.register(GameCreateAttemptPacket.class);
        k.register(GameCreateResponsePacket.class);
        k.register(GameListGetPacket.class);
        k.register(GameListSendPacket.class);
        k.register(GameLevel.class);
        k.register(GameLevelEnum.class);
        k.register(ArrayList.class);

        server.bind(PORT);
        server.start();
        server.addListener(new KryonetExampleListener());
    }
}
```

Výpis 5: Vytvoření instance serveru v Kryonetu

```
public class KryonetClientExample
{
    public void startClient() throws IOException
    {
        private Client client;

        private static int PORT = 54345;
        private static String HOST = "127.0.0.1";
        private static int TIMEOUT = 60000;

        client = new Client();
        Kryo k~= client.getKryo();
        k.register(PlayerGameEnterAttemptPacket.class);
        k.register(PlayerGameEnterResponsePacket.class);
        k.register(GameCreateAttemptPacket.class);
        k.register(GameCreateResponsePacket.class);
        k.register(GameListGetPacket.class);
        k.register(GameListSendPacket.class);
        k.register(GameLevel.class);
        k.register(GameLevelEnum.class);
        k.register(ArrayList.class);

        client.start();
        client.connect(TIMEOUT, HOST, PORT);
    }
}
```

Výpis 6: Vytvoření instance klienta v Kryonetu

3.7 XML

XML [5] je značkovací jazyk podobný jazyku HTML sloužící pro popis datové struktury a přenos dat. XML je navrženo s důrazem na strojovou čitelnost, ale také přehlednost pro

```
<?xml version="1.0" encoding="utf-8"?>
<catalog>
	<product>
		<product_url>http://www.mywebsite.com/products/TV-LCD-THOMSON-32HE8234B.html</product_url>
		<designation>TV LCD THOMSON 32HE8234B</designation>
		<price>369.00</price>
		<category>Home/Television/LCD TVs/Thomson</category>
		<image_url>http://www.mywebsite.com/products/1/32/324805268_F.jpg</image_url>
		<description>Format 16/9 screen size 32" [...] 2 bass reflex speakers</description>
		<brand>Thomson</brand>
		<merchant_id>2148</merchant_id>
		<manufacturer_id>THL324805268</manufacturer_id>
		<shipping_cost>0</shipping_cost>
		<in_stock>Y</in_stock>
		<stock_detail>49</stock_detail>
		<condition>0</condition>
		<isbn></isbn>
		<upc_ean>324805268</upc_ean>
		<product_type>1</product_type>
	</product>
	<product>
		<product_url>http://www.mywebsite.com/books/The-Third-Policeman-9781564782144.html</product_url>
		<designation>The Third Policeman</designation>
		<price>10.04</price>
		<category>Home/Books/Fiction Books/Fiction/Literature</category>
		<image_url>http://www.mywebsite.com/books/4/12/9781564782144B.jpg</image_url>
		<description>Originally submitted and rejected [...] after the author`s death.</description>
		<brand>Naxos</brand>
		<merchant_id>BK9781564782144</merchant_id>
		<manufacturer_id>9781564782144</manufacturer_id>
		<shipping_cost>3.99</shipping_cost>
		<in_stock>Y</in_stock>
		<stock_detail>3</stock_detail>
		<condition>1</condition>
		<isbn>9781564782144</isbn>
		<upc_ean></upc_ean>
		<product_type>4</product_type>
	</product>
</catalog>
```

Obrázek 2: Ukázka XML dokumentu

uživatele.

XML je otevřený formát dat, který navrhlo konsorcium W3C již v roce 1996. Své první verze se dočkal o rok později v roce 1997, označované Extensible Markup Language 1.0. Od té doby je verze 1.0 postupně vyvíjena. Poslední úprava této verze proběhla v roce 2008.

V roce 2004 byl publikován popis XML 1.1, který řešil drobné nedostatky verze 1.0 a to především podpora pro systémy EBCDIC a zároveň začal podporovat znaky nepatřící do sady Unicode 3.2. Jelikož změny jsou důležité pouze pro specifická použití, XML 1.1 není velmi rozšířen a podporován. Poslední úprava tohoto standardu proběhla v roce 2006.

Struktura XML je stromová a skládá se z tagů, které mohou nabývat určitých hodnot. Zároveň každý tag může mít přiřazeny své atributy a ve svém těle obsahovat další tagy. Tag na nejvyšší úrovni se nazývá kořen.

Strukturu dokumentu lze nadefinovat pomocí jazyka XSD. Ukázkový XML dokument je vyobrazen na obrázku 2.

3.8 Simple XML

Simple [3] je serializační knihovna pro jazyk XML, kterou lze využít jak v desktopové, tak v Android aplikaci. Zvládá práci s cyklickým grafem objektů, a tak dokáže serializovat i složitou strukturu objektů. Na rozdíl od svých konkurentů nepotřebuje žádnou konfiguraci.

Třídu, kterou chceme serializovat označíme anotací `@Root`, která přijímá parametr `name`, který ovlivňuje název kořenového prvku XML souboru. Vlastnosti, které chceme serializovat jako potomky daného prvku označíme anotací `@Element`. Pokud chceme k aktuálnímu prvku přidat vlastnost jako atribut, označíme takovou vlastnost anotací `@Attribute`.

```
@Root
public class Example {

   @Element
   private String  text ;

   @Attribute
   private int  index;

   public Example() {
      super();
   }

   public Example(String text, int  index)  {
      this . text  = text ;
      this .index = index;
   }

   public String  getMessage() {
      return text ;
   }

   public int  getId ()  {
      return index;
   }
}
```

Výpis 7: Ukázka serializace objektu pomocí serializátoru Simple

4 Praktická část

V této části se budeme zabývat praktickým vývojem aplikace a postupy, které byly použity při její tvorbě. Celá aplikace se dělí na čtyři podčásti. Herní knihovnu FreedomLibrary, implementaci desktop verze FreedomGame, implementaci mobilní verze hry pro platformu Android FreedomAndroid a implementaci serveru FreedomServer.

4.1 FreedomLibrary

FreedomLibrary je základní knihovnou, která obsahuje všechny potřebné třídy pro napojení na GUI. Cílem této knihovny je co nejvíce zjednodušit tvorbu grafických rozhraní pro dekstop a Android platformy.

4.1.1 Game

Instance třídy představuje aktuální stav každé hry. Jednotlivé stavy jsou popsány v tabulce 3.

Hra obsahuje informaci o jejím názvu, počtu hráčů, určuje rozmezí generátoru pohybu multiplerů pro akciové karty, obsahuje pole všech hratelných karet a hráčů.

Třída `Game` obsahuje pouze informace o stavu hry. Samotné chování je řešeno v třídě `GameBehavior`. Ta umožňuje ovlivnit chování aplikace v reakci na různé události a obsahuje metody pro ovládání stavu hry `start()`, `pause()` a `stop()`.

4.1.2 Player

Třída hráče obsahuje informace o daném hráči, jeho jméno a přiděleného klienta. Třída `Player` má potomky `HumanPlayer` a `ComputerPlayer`. Implementují metodu `play()`, která je volána každé hrací kolo a obsahuje všechny důležité události, které se musí při začátku nového kola stát, tedy úprava finanční hotovosti klienta v závislosti na daném cashflow, kontrola finanční situace klienta a případné přidání karty půjčky, odehrání všech karet aktivit a táhnutí karty události z balíčku.

Třída `ComputerPlayer` navíc implementuje základní umělou inteligenci, která implementuje agresivní vítěznou strategii. Preferováno je splacení dluhů a poté rychlý nákup akcií.

Závislosti jednotlivých tříd hlouběji popisuje Class diagram na obrázku číslo 3.

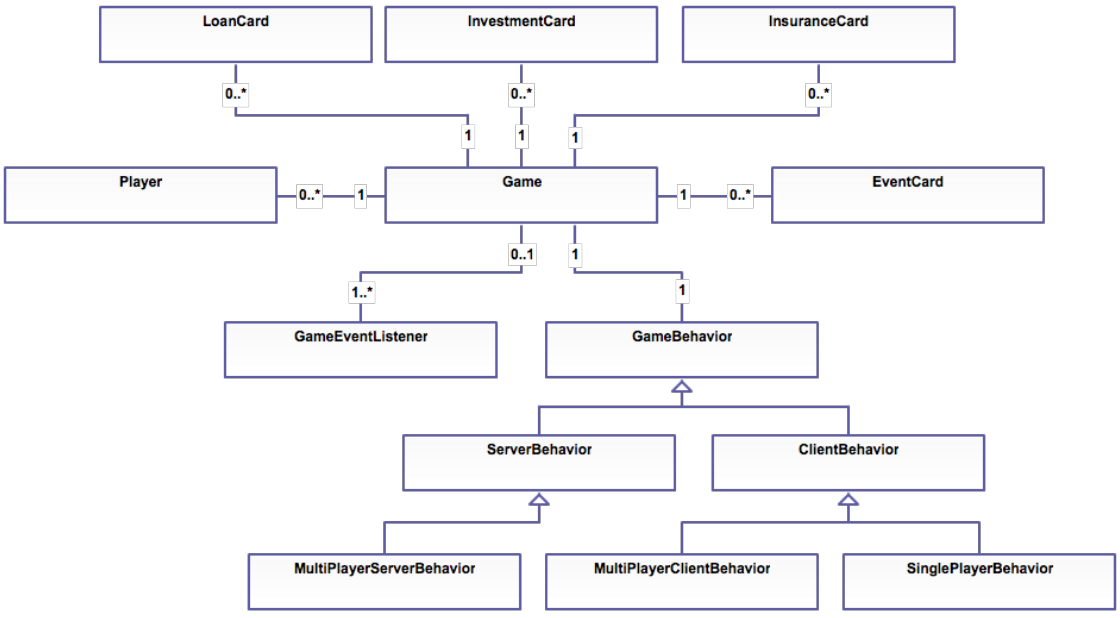
4.1.3 Family

Třída rodiny obsahuje informace o dané rodině, jméno, členy rodiny, cíle rodiny a požadovanou finanční nezávislost, aktuální finanční hotovost, výdaje a dluhy rodiny.

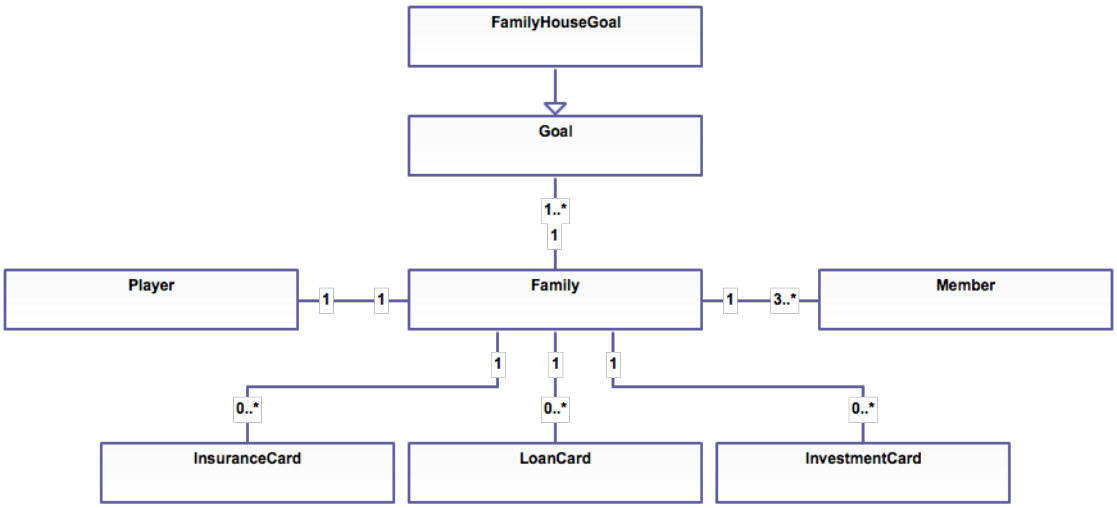
Závislosti této třídy podrobněji popisuje Class diagram na obrázku číslo 4.

Stav hry	Popis stavu
GAME_INITIATED	Základní stav hry, instance byla vytvořena
GAME_WAITING_FOR_PLAYERS	Do hry není připojen dostatečný počet hráčů
GAME_RUNNING	Hra právě probíhá
GAME_PAUSED	Hra byla pozastavena včetně časovače kola
GAME_STOPPED	Hra byla přerušena a již nemůže být obnovena
GAME_FINISHED	Hra byla odehrána, je znám vítěz
GAME_WAITING_FOR_RESPONSE	Hra čeká na odpověď serveru nebo klienta

Tabulka 3: Jednotlivé stavy hry



Obrázek 3: Class Diagram třídy Game



Obrázek 4: Class Diagram třídy Family

4.1.4 Activity Card

Třída `ActivityCard` je obecnou implementací karty, kterou může daný hráč nakoupit či prodat a ovlivňuje finanční situaci hráče. Potomek třídy ovlivňuje chování karty pomocí metod `activate()`, `decativate()` a `play()`.

Metoda `activate()` je volána v době nákupu dané karty a jako parametr přijímá objekt rodiny, která kartu kupuje. V této metodě můžeme ovlivnit například snížení finanční hotovosti klienta v závislosti na ceně karty. Pro přidání další funkcionality při nákupu karty lze tuto metodu přepsat u potomků této třídy.

Metoda `deactivate()` je volána v době prodeje dané karty a jako parametr přijímá objekt rodiny, která kartu prodává. V této metodě můžeme ovlivnit například cenu za prodej karty, vypořádání se s úroky za předčasný prodej karty či vyvolat výjimku, která zabraňuje kartu prodat.

Metoda `play()` je volána jednou v každém kole hry. V této metodě můžeme ovlivnit například snížení finanční hotovosti klienta v závislosti na počtu splátek. Chování této metody je dále upravena u potomků této třídy.

Metoda `canBePlayed()` slouží k zjištění, zda je možné v daném okamžiku danou kartu odehrát. Metoda `canBeActivated()` slouží k zjištění, zda je možné v daném okamžiku karta aktivována. Obdobná metoda `canBeDeactivated()` slouží k zjištění, zda je možné v daném okamžiku karta deaktivována.

Metoda `getActivationPrize()` slouží k zjištění množství peněz, které jsou potřeba k aktivaci karty. Metoda `getDeactivationPrize()` slouží k zjištění množství peněz, které jsou potřeba k deaktivaci karty. Metoda `getInstallmentPrize()` slouží k zjištění množství peněz, které jsou potřeba k odehrání karty.

Všechny metody přijímají jako parametr třídu Family a jsou abstraktní. Proto je musí implementovat jeden z potomků.

Závislosti třídy `ActivityCard` jsou znázorněny na obrázku 5.

4.1.5 Event Card

Třída `EventCard` je obecnou implementací karty, kterou si hráč tahá z balíku karet. Karta může ovlivnit jeho finanční situaci. Metoda `play()` je volána v době, kdy si hráč kartu vytáhne z balíčku karet. Tato metoda určuje chování karty, tedy například úmrtí v rodině nebo narození potomka. Každý potomek třídy `EventCard` musí tuto metodu implementovat.

4.1.6 Levely hry

Hra Finanční Svoboda obsahuje různé druhy obtížnosti. Ty se liší především finanční situací klientů a její náročností na vyřešení a výběr správného postupu k dosažení finanční nezávislosti. Úrovně hry jsou popsány pomocí jazyka XML.

Hra obsahuje v balíčku `freedom.util` vlastní serializátor těchto úrovní, který obsahuje dvě statické metody `in()` a `out()`. Pomocí těchto metod je možné načítat a ukládat úrovně hry z a do XML. Návratovou hodnotou funkce `in()` je objekt třídy `Game`.

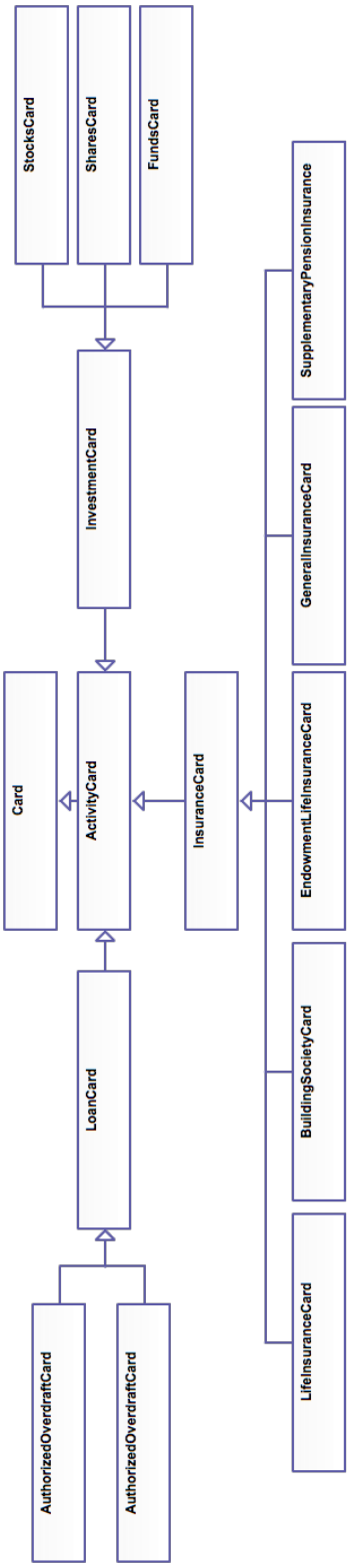
4.1.7 Rozšiřitelnost hry

Hra je implementována s důrazem na možnost pozdějšího rozšíření obsahu nebo herní logiky třemi způsoby.

Rozšíření hry je implementováno pomocí úrovní. Každá úroveň je popsána jako samostatný XML soubor. Ukázku XML souboru úrovně najdete v příloze B.

```
<investmentCards>
    <investmentCard class="freedom.model.card.activity.FundsCard" />
    <investmentCard class="freedom.model.card.activity.SharesCard" />
    <!-- vlastni karta -->
    <investmentCard class="freedom.model.card.activity.GoldenCard" />
</investmentCards>
<loanCards>
    <loanCard class="freedom.model.card.activity.MordgageCard" />
    <loanCard class="freedom.model.card.activity.AuthorizedOverdraftCard" />
    <!-- vlastni karta -->
    <loanCard class="freedom.model.card.activity.BankRobberyCard" />
</loanCards>
<insuranceCards>
    <insuranceCard class="freedom.model.card.activity.AssetInsuranceCard" />
    <insuranceCard class="freedom.model.card.activity.BuildingSocietyCard" />
    <!-- vlastni karta -->
    <insuranceCard class="freedom.model.card.activity.InprisonmentInsuranceCard" />
</insuranceCards>
```

Výpis 8: Tvorba vlastní úrovně hry



Obrázek 5: Class Diagram třídy ActivityCard

Můžeme napsat vlastní třídu karty aktivity, která musí být potomkem jedné ze tříd `InsuranceCard`, `InvestmentCard` nebo `LoanCard`, které jsou dále potomkem třídy `ActivityCard` a implementuje metodu `play()`, která jako atribut přijímá instanci třídy `Family`. Tato karta bude automaticky přidána do grafického rozhraní aplikace. Je ale nutné takovou kartu zaregistrovat do seznamu hratelných karet v XML souboru úrovně. Přidáváme pouze jeden objekt třídy Class.

Dále je možné implementovat vlastní kartu události, která musí být potomkem třídy `EventCard` a implementovat metodu `play()`, která jako parametr přijímá instanci třídy `Family`. Tato karta bude automaticky přidána do balíku karet události, ze kterého si hráči tahají karty každé kolo hry. Kartu je nutné přidat do XML reprezentace úrovně. Jednu kartu můžete přidat i vícekrát.

Pro úpravu chování hry je možné rozšířit třídy `ClientBehavior` a `ServerBehavior`. Upravit lze veškeré chování hry, včetně délky jednoho kola, techniky generování vývoje ceny prvků finančního trhu či tažení karet události. Seznam metod je podrobněji rozepsán v tabulce 4.

4.2 FreedomServer

Freedom Server je server hry Finanční Svoboda, který v sobě implementuje knihovnu `FreedomLib`. Pro komunikaci mezi serverem a klientem využívá knihovnu KryoNet.

4.2.1 Síťová komunikace

KryoNet podporuje vytvoření síťové komunikace pomocí TCP i UDP. Vzhledem k charakteru hry preferujeme spolehlivý přenos informací před rychlostí, a proto byl jako komunikační protokol zvolen TCP.

Připojení v KryoNetu musí mít nastavenou hodnotu TIMEOUT. Po uplynutí této doby dojde k automatickému ukončení spojení. Dá se ovšem předpokládat, že se někteří hráči do hry pár kol nezapojí, a proto je mezi serverem a klientem pravidelně posílán `PingPacket`, který ověřuje aktivní spojení mezi klientem a serverem a naopak.

Mezi klientem a serverem jsou posílány pakety blíže znázorněné v tabulce 5.

Hra je schopna při zjištění nevalidity dat na straně klienta hru znovu synchronizovat posláním celé třídy `Game`. V takovém případě je na straně klienta vytvořen paket `GameSyncRequestPacket` a čeká se na odpověď v podobě paketu `GameSyncPacket`.

4.2.2 Propojení s FreedomLibrary

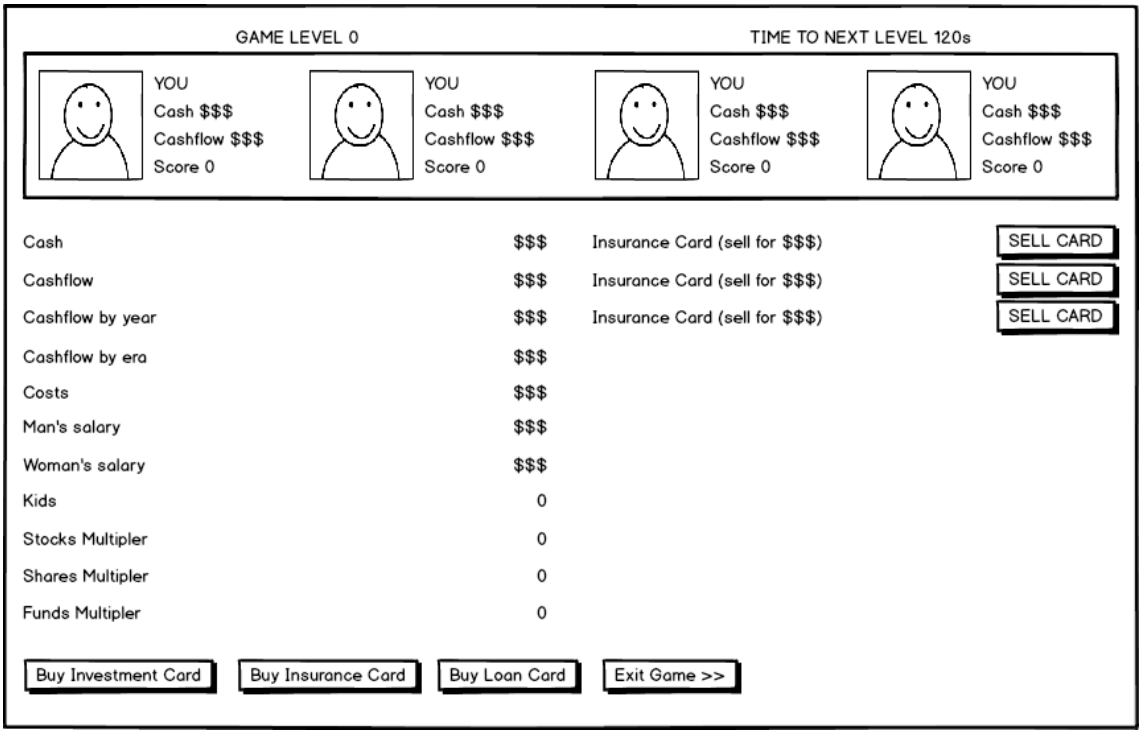
Pro vytvoření seznamu her využijeme třídu `Lobby`, která obsahuje kolekci tříd `Game`, která reprezentuje aktuální stav hry. Pro správnou funkčnost hry je nutné pomocí metody `setBehavior()` nastavit chování hry. Součástí herní knihovny FreedomLibrary je serverové chování hry `MultiplayerServerBehavior`, potomek třídy `Behavior`.

Metoda	Popis paketu
`onStart()`	Volána při začátku hry
`onPause()`	Volána při přerušení hry
`onStop()`	Volána při zastavení hry
`onRestart()`	Volána při restartu hry
`onNextLevel()`	Volána při začátku nového kola hry
`onMultiplersChanged()`	Volána při změně multiplerů, probíhá s voláním `onNextLevel()`
`onActivityCardAdded()`	Volána při nákupu karty hráčem
`onActivityCardRemoved()`	Volána při prodeji karty hráčem
`onActivityCardPlayed()`	Volána při odehrání karty
`onEventCardPlayed()`	Volána při vytáhnutí karty události z balíku
`onPlayerJoined()`	Volána při přidání hráče do hry
`onPlayerLeft()`	Volána při odpojení hráče ze hry

Tabulka 4: Popis metod třídy Behavior

Typ paketu	Popis paketu	Odesílán klientem	Odesílán serverem
`GameCreateAttemptPacket`	Požadavek o vytvoření hry	ANO	NE
`GameCreateResponsePacket`	Odpověď k vytvoření hry	NE	ANO
`GameListGetPacket`	Požadavek o seznam her	ANO	NE
`GameListSendPacket`	Odpověď se seznamem her	NE	ANO
`GamePlayerActivityCardAdded`	Hráč koupil kartu aktivit	ANO	NE
`GamePlayerActivityCardRemoved`	Hráč prodal kartu aktivit	ANO	NE
`GamePlayerEventCardPlayed`	Odeslání karty klient hráči	NE	ANO
`GamePlayerJoinedPacket`	Do hry se připojil nový hráč	NE	ANO
`GamePlayerExitPacket`	Hráč se manuálně odpojil	NE	ANO
`GamePlayerTimeoutPacket`	Platnost spojení vypršela	NE	ANO
`GameSyncPacket`	Synchronizační packet	ANO	ANO
`GameSyncRequestPacket`	Požadavek o synchronizaci hry	ANO	ANO
`PingPacket`	Ping opačné strany	ANO	ANO
`PongPacket`	Odpověď na dotaz ping	ANO	ANO

Tabulka 5: Jednotlivé stavy hry



Obrázek 6: Návrh grafického rozhraní pro desktop

4.3 FreedomGame

V následující kapitole si popíšeme průběh implementace desktopové verze hry Finanční svoboda.

4.3.1 Tvorba GUI

Grafické rozhraní ve hře je tvořeno pomocí knihovny `Nifty-GUI`. Při navrhování GUI byl kladen důraz na jednoduchost a přehlednost tak, aby se zvyšovala hratelnost hry.

GUI bylo rozděleno do čtyř horizontálních zón. V první zóně se nachází informace o dané hře, aktuální kolo a zbývající čas do dalšího kola. Ve druhé části je zobrazen seznam hráčů a základní informace o stavu jejich klienta a počtu bodů, které byli schopni získat. V levé části třetí zóny jsou vyobrazeny detailní informace o klientovi hráče a vývoj na finančním trhu. V pravé části jsou všechny aktuální karty klienta a možnost karty prodat. V nejspodnější části rozhraní jsou vyobrazeny tlačítka pro nákup karet a opuštění hry. Návrh GUI je zobrazen na obrázku 6.

4.3.2 Propojení s FreedomLibrary

Pro propojení s FreedomLibrary využijeme připravenou třídu `Game`, která reprezentuje aktuální stav hry, obsahuje pole všech hráčů, balíček karet událostí. Pro správnou funkč-

nost třídy musíme metodou `setBehavior` nastavit chování hry. Tato metoda přijímá jako parametr instanci třídy `Behavior`. Součástí knihovny jsou tři základní chování hry. Třída `SinglePlayerBehavior` obsahuje základní chování hry jednoho hráče a je potomkem třídy `ClientBehavior`. Třída `MultiplayerClientBehavior` obsahuje základní chování klienta ve hře více hráčů a je také potomkem třídy `ClientBehavior`.

Pro správnou funkčnost herních událostí, je třeba implementovat v aplikaci EventListener [9], který implementuje rozhraní `GameEventListener`.

Pro vytvoření předpřipravené instance je možné využít statických metod třídy `Game` `getSinglePlayerWithData` či `getMultiplayerClientWithData`. Obě tyto metody přijímají jako parametr objekt třídy `GameEvent`.

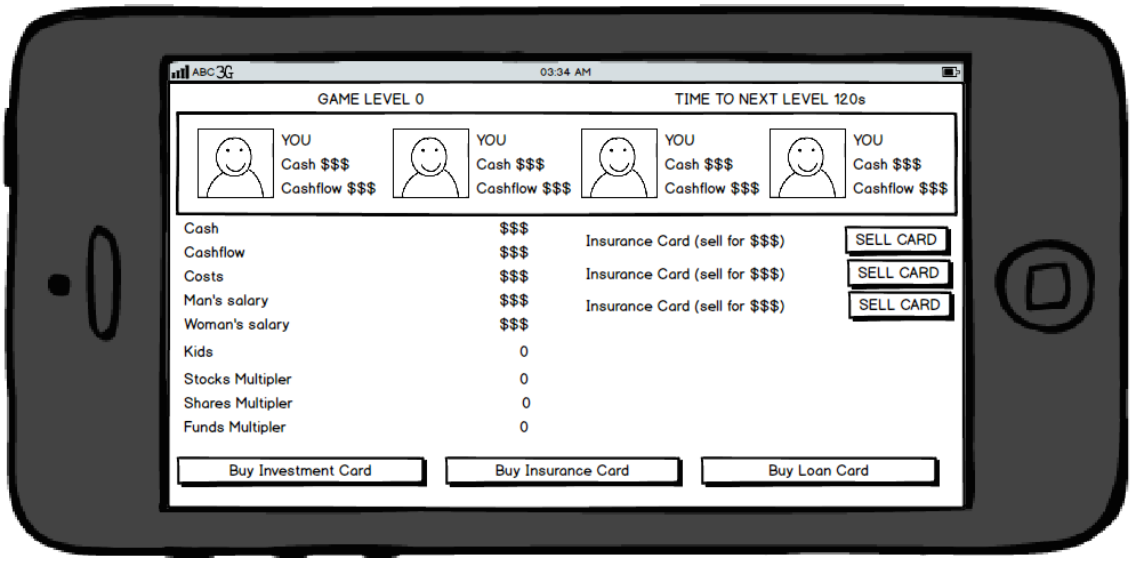
V případě hry více her se používá připravená třída `SingletonClient`, která realizuje návrhový vzor Singleton [1] a zajišťuje propojení se serverem na dané IP adrese, dokáže zjistit pomocí metody `hasConnectionToServer()` zda je definovaný server aktivní a zajistit základní komunikaci mezi serverem a klientem.

4.4 FreedomAndroid

V následující sekci si popíšeme průběh implementace mobilní verze hry Finanční svoboda.

4.4.1 Úprava GUI pro mobilní zařízení

Pokusíme se zachovat stejné rozložení zón jako na desktopu, nicméně počet informací zmenšíme. Z grafického rozhraní nejprve odstraníme nadbytečné informace Cashflow by Year a Cashflow by Era. Z dolní lišty odstraníme tlačítko zpět, jelikož to je zakomponováno přímo v sytému Android a nemá tedy žádnou funkci. Návrh rozhraní pro mobilní telefony Android je vyobrazen na obrázku 7.



Obrázek 7: Návrh grafického rozhraní pro Android

5 Závěr

Cílem této bakalářské práce bylo navrhnout a implementovat elektronickou verzi deskové hry Finanční svoboda pro desktop a mobilní platformu Android.

Celý systém byl navržen tak, že jeho implementace byla rozdělena do co nejmenších celků, které mezi sebou komunikují. Vyhneme se tak problémům s úpravami většího množství kódu i při provádění menších změn.

Bylo nutné volit vhodné technologie, které podporují desktopovou verzi jazyka Java i mobilní verzi pro systém Android. Nedostupnost některých balíčků na platformě Android byla klíčová při vývoji knihovny. Původně zamýšlený serializační balíček JAXB musel být nahrazen balíčkem Simple XML, jelikož JAXB není pro platformu Android implementován kvůli své paměťové náročnosti, která pro daná zařízení není vhodná.

Při vývoji serverové části aplikace jsem narazil na problém serializace objektů s cyklickými závislostmi. Serializační knihovna Kryo používaná komunikačním frameworkem KryoNet ignorovala anotaci `@Transient` přesto, že v dokumentaci frameworku je uveden přesný opak. Problém byl vyřešen použitím speciální anotace, kterou používá serializátor knihovny KryoNet.

Vzhledem k tomu, že Java herní server vyvíjený současně s touto bakalářskou prací nebyl vyvinut, není v aplikaci implementován.

V Android aplikaci bylo GUI upraveno tak, aby hra byla na zařízeních s menší obrazovkou co nejlépe hratelná, přesto aby si zachovala rysy plnohodnotné desktopové aplikace. Při synchronizaci větších objektů mobilní klient automaticky uzavírá spojení se serverem a hra se stává nekonzistentní. U připojení dvou desktopových aplikací tento problém nenastává.

Vývoj aplikace důkladně prověřil mé programátorské schopnosti a prohloubil znalosti jazyka Java a vývoje aplikace her pro mobilní platformu Android a také schopnost abstrahovat dané problémy.

6 Reference

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four), Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025

[2] Darwin, Ian F., Java Cookbook. 2nd ed. Sebastopol,CA: O'Reilly, c2004, xxiv, 829 p. ISBN 05-960-0701-9. Dostupné z: http://it-ebooks.info/book/2249/

[3] Using Simple for XML serialization. [online]. [cit. 2014-04-20]. Dostupné z: http://www.ibm.com/developerworks/java/library/x-simplexobjs/index.html?ca=dgr-twtrSimple2XMLdth-XML

[4] Java (programming lanuage). [online]. [cit. 2014-04-20]. Dostupné z: http://en.wikipedia.org/wiki/Java_(programming_language)

[5] XML. [online]. [cit. 2014-04-20]. Dostupné z: http://en.wikipedia.org/wiki/XML

[6] About LWJGL. [online]. [cit. 2014-04-20]. Dostupné z: http://lwjgl.org/wiki/index.php?title=About_LWJGL

[7] OpenGL. [online]. [cit. 2014-04-20]. Dostupné z: http://en.wikipedia.org/wiki/OpenGL

[8] Nifty Standard Controls. [online]. [cit. 2014-04-20]. Dostupné z: http://sourceforge.net/apps/mediawiki/nifty-gui/index.php?title=Nifty_Standard_Controls_(Nifty_1.3)

[9] Lessons: Writing Event Listeners. [online]. [cit. 2014-04-20]. Dostupné z: http://docs.oracle.com/javase/tutorial/uiswing/events/

[10] TCP and UDP client/server library for Java. [online]. [cit. 2014-05-07]. Dostupné z: https://github.com/EsotericSoftware/kryonet

[11] Java serialization and cloning: fast, efficient, automatic. [online]. [cit. 2014-05-07]. Dostupné z: https://github.com/EsotericSoftware/kryonet

A Obsah CD

Přiložené CD obsahuje následující soubory:

- Text této bakalářské práce ve formátu PDF
- Ukázkový XML soubor úrovně
- Spustitelnou verzi programu
- Zdrojové kódy programu

B Ukázkový XML soubor úrovně

```
<game>
    <randomMin>7</randomMin>
    <randomMax>−3</randomMax>
    <playersLimit>4</playersLimit>
    <stocksMultipler>0</stocksMultipler>
    <sharesMultipler>0</sharesMultipler>
    <fundsMultipler>0</fundsMultipler>
    <families>
        <family>
            <name>Brzobohati</name>
            <man>
                <name>Bedrich</name>
                <gender>1</gender>
                <salary>13500</salary>
                <alive>true</alive>
            </man>
            <woman>
                <name>Bozena</name>
                <gender>2</gender>
                <salary>10000</salary>
                <alive>true</alive>
            </woman>
            <kids>
                <member>
                    <name>Bara</name>
                    <gender>4</gender>
                    <salary>0</salary>
                    <alive>true</alive>
                </member>
            </kids>
            <goals>
                <goal class="freedom.model.goal.FamilyHouseGoal">
                    <name>Byt</name>
                    <description>Byt pro celou rodinu</description>
                    <cost>2500000</cost>
                    <threshold>3</threshold>
                </goal>
                <goal class="freedom.model.goal.Goal">
                    <name>Automobil</name>
                    <description>Automobil 123</description>
                    <cost>400000</cost>
                    <threshold>6</threshold>
                </goal>
            </goals>
            <cash>50000</cash>
            <houseRental>6750</houseRental>
            <financialIndependence>100000000</financialIndependence>
            <expenses>20000</expenses>
            <insuranceCards/>
            <loanCards>
                <loanCard class="freedom.model.card.activity.AuthorizedOverdraftCard">
```

```
            <active>false</active>
            <playedCount>0</playedCount>
        </loanCard>
        <loanCard class="freedom.model.card.activity.AuthorizedOverdraftCard">
            <active>false</active>
            <playedCount>0</playedCount>
        </loanCard>
        <loanCard class="freedom.model.card.activity.AuthorizedOverdraftCard">
            <active>false</active>
            <playedCount>0</playedCount>
        </loanCard>
    </loanCards>
    <investmentCards/>
</family>
<family>
    <name>Sporivi</name>
    <man>
        <name>Sebastian</name>
        <gender>1</gender>
        <salary>20000</salary>
        <alive>true</alive>
    </man>
    <woman>
        <name>Sona</name>
        <gender>2</gender>
        <salary>15250</salary>
        <alive>true</alive>
    </woman>
    <kids>
        <member>
            <name>Sara</name>
            <gender>4</gender>
            <salary>0</salary>
            <alive>true</alive>
        </member>
    </kids>
    <goals>
        <goal class="freedom.model.goal.FamilyHouseGoal">
            <name>Byt</name>
            <description>Byt pro celou rodinu</description>
            <cost>2500000</cost>
            <threshold>3</threshold>
        </goal>
        <goal class="freedom.model.goal.Goal">
            <name>Automobil</name>
            <description>Automobil 123</description>
            <cost>400000</cost>
            <threshold>6</threshold>
        </goal>
    </goals>
    <cash>25000</cash>
    <houseRental>9000</houseRental>
    <financialIndependence>3500000</financialIndependence>
    <expenses>15000</expenses>
```

```
    <insuranceCards/>
    <loanCards>
        <loanCard class="freedom.model.card.activity.AuthorizedOverdraftCard">
            <active>false</active>
            <playedCount>0</playedCount>
        </loanCard>
        <loanCard class="freedom.model.card.activity.AuthorizedOverdraftCard">
            <active>false</active>
            <playedCount>0</playedCount>
        </loanCard>
        <loanCard class="freedom.model.card.activity.AuthorizedOverdraftCard">
            <active>false</active>
            <playedCount>0</playedCount>
        </loanCard>
    </loanCards>
    <investmentCards/>
</family>
<family>
    <name>Korunkovi</name>
    <man>
        <name>Karel</name>
        <gender>1</gender>
        <salary>20750</salary>
        <alive>true</alive>
    </man>
    <woman>
        <name>Kamila</name>
        <gender>2</gender>
        <salary>11500</salary>
        <alive>true</alive>
    </woman>
    <kids>
        <member>
            <name>Kristyna</name>
            <gender>4</gender>
            <salary>0</salary>
            <alive>true</alive>
        </member>
    </kids>
    <goals>
        <goal class="freedom.model.goal.FamilyHouseGoal">
            <name>Byt</name>
            <description>Byt pro celou rodinu</description>
            <cost>2500000</cost>
            <threshold>3</threshold>
        </goal>
        <goal class="freedom.model.goal.Goal">
            <name>Automobil</name>
            <description>Automobil 123</description>
            <cost>400000</cost>
            <threshold>6</threshold>
        </goal>
    </goals>
    <cash>70000</cash>
```

```
    <houseRental>9000</houseRental>
    <financialIndependence>3000000</financialIndependence>
    <expenses>12000</expenses>
    <insuranceCards/>
    <loanCards>
        <loanCard class="freedom.model.card.activity.AuthorizedOverdraftCard">
            <active>false</active>
            <playedCount>0</playedCount>
        </loanCard>
        <loanCard class="freedom.model.card.activity.AuthorizedOverdraftCard">
            <active>false</active>
            <playedCount>0</playedCount>
        </loanCard>
        <loanCard class="freedom.model.card.activity.AuthorizedOverdraftCard">
            <active>false</active>
            <playedCount>0</playedCount>
        </loanCard>
    </loanCards>
    <investmentCards/>
</family>
<family>
    <name>Moudri</name>
    <man>
        <name>Mojmir</name>
        <gender>1</gender>
        <salary>23000</salary>
        <alive>true</alive>
    </man>
    <woman>
        <name>Monika</name>
        <gender>2</gender>
        <salary>10250</salary>
        <alive>true</alive>
    </woman>
    <kids>
        <member>
            <name>Marketa</name>
            <gender>4</gender>
            <salary>0</salary>
            <alive>true</alive>
        </member>
    </kids>
    <goals>
        <goal class="freedom.model.goal.FamilyHouseGoal">
            <name>Byt</name>
            <description>Byt pro celou rodinu</description>
            <cost>2500000</cost>
            <threshold>3</threshold>
        </goal>
        <goal class="freedom.model.goal.Goal">
            <name>Automobil</name>
            <description>Automobil 123</description>
            <cost>400000</cost>
            <threshold>6</threshold>
```

```
            </goal>
        </goals>
        <cash>90000</cash>
        <houseRental>8500</houseRental>
        <financialIndependence>3000000</financialIndependence>
        <expenses>15000</expenses>
        <insuranceCards/>
        <loanCards>
            <loanCard class="freedom.model.card.activity.AuthorizedOverdraftCard">
                <active>false</active>
                <playedCount>0</playedCount>
            </loanCard>
            <loanCard class="freedom.model.card.activity.AuthorizedOverdraftCard">
                <active>false</active>
                <playedCount>0</playedCount>
            </loanCard>
            <loanCard class="freedom.model.card.activity.AuthorizedOverdraftCard">
                <active>false</active>
                <playedCount>0</playedCount>
            </loanCard>
        </loanCards>
        <investmentCards/>
    </family>
</families>
<investmentCards>
        <investmentCard class="freedom.model.card.activity.FundsCard" />
        <investmentCard class="freedom.model.card.activity.SharesCard" />
        <investmentCard class="freedom.model.card.activity.StocksCard" />
 </investmentCards>
 <loanCards>
        <loanCard class="freedom.model.card.activity.MordgageCard" />
        <loanCard class="freedom.model.card.activity.AuthorizedOverdraftCard" />
 </loanCards>
 <insuranceCards>
        <insuranceCard class="freedom.model.card.activity.AssetInsuranceCard" />
        <insuranceCard class="freedom.model.card.activity.BuildingSocietyCard" />
        <insuranceCard class="freedom.model.card.activity.EndowmentLifeInsuranceCard" />
        <insuranceCard class="freedom.model.card.activity.GeneralInsuranceCard" />
        <insuranceCard class="freedom.model.card.activity.LifeInsuranceCard" />
        <insuranceCard class="freedom.model.card.activity.SupplementaryPensionInsuranceCard"
            />
 </insuranceCards>
 <eventCards>
    <eventCard class="freedom.model.card.event.MoneySpentEventCard">
        <id>0</id>
        <name>Squash</name>
        <description>Vas klient  se rozhodl koupit  vysavac v~celkove cene 38 000</description>
        <cost>38000</cost>
    </eventCard>
    <eventCard class="freedom.model.card.event.ChildBornEventCard">
        <id>0</id>
        <name>Dite</name>
        <description>Vasim klientum se narodilo dite</description>
    </eventCard>
```

```
    </eventCards>
</game>
```

Výpis 9: Ukázkový XML soubor úrovně